LOG

The Chipmunk Logic Simulator

By David Gillespie

USER'S GUIDE

5130:TR:84

Computer Science
California Institute of Technology
Pasadena CA 91125

Table of Contents

Chapter I

Introduction

Introduction to *LOG*

LOG is a program for designing and simulating digital logic circuits. Using the graphics tablet and pen, the user draws gates and wires onto the screen to form a conventional schematic diagram. Special symbols are used to represent switches, LED's, and other devices connected to the circuit. The computer continuously simulates the state of the circuit based on the settings of the switches connected. Because of its highly interactive approach, *LOG* is quickly learned and easy to use.

Features of *LOG* include on-line Help, Scroll and Zoom for working with large circuits, a variety of predefined gates and devices (including a large library of TTL 7400's series parts), and advanced editing commands. Basic operations, such as adding, moving, and deleting objects are simple and intuitive.

LOG is most suitable for the simulation of simple circuits. Students learning logic design basics would find *LOG* to be a perfect way to try out their ideas. *LOG* is fast enough to simulate moderately-sized circuits interactively; as circuit size increases, cursor motion becomes too slow for interactive simulation, but *LOG* allows simulation to be turned off when it is not needed.

Programs Used by *LOG*

The following programs and files are used by the Logic Simulator:

LOG.CODE	This is the main program for the Logic Simulator. It is the program which allows you to edit and simulate a circuit.
GATES	This file contains the Gate Library. For every kind of gate it describes how the gate looks and how it responds to incoming signals. The *LOGED* program is used to add, remove, or change the descriptions of gates.
GATESIX	This file contains an index into the GATES file. It is automatically created and updated by *LOGED*.
LOGHELP.TEXT	This is the Help file. It contains the text screens which are displayed when Help is requested. CAGED or another text editor may be used to edit this file.

LOGHELPIX	This is an index file for LOGHELP.TEXT. It is created and updated by the *LOGH* program. LOGHELP.TEXT and LOGHELPIX are not required, but the Help command will not work if these files are absent.
LOGED.CODE	This program is used to edit the GATES file. It is described in chapter IV of this manual.
LOGH.CODE	This program updates the index file after changes have been made to LOGHELP.TEXT. It is described in chapter V.

Hardware Required by *LOG*

The Logic Simulator currently runs only on HP 9836C computers under the Pascal 2.0 operating system. The only peripheral required is the 9111A digitizing tablet.

Peripherals Used By *LOG*

LOG on the 9836C takes advantage of all I/O devices. These include the graphics and text screens, the digitizing tablet, the keyboard, the built-in beeper, and the printer.

Color graphics screen. Ordinarily, most of the screen is available for displaying circuits. At the bottom of the screen is the *menu area* containing the words SAVE, LOAD, AUTO, HELP, CAT, DEL, and ON or OFF. Pressing the pen over any of these words selects the corresponding feature. Also at the bottom of the screen is a selection of the most commonly used gates. Pressing CAT displays the full catalog of gates on the screen.

Tablet and Pen. When the pen is held near the surface of the tablet, a *cursor* appears in the corresponding place on the color screen. The cursor is used to point to objects and parts of the screen. Three basic pen movements are used in controlling *LOG*. *Pulling* an object means pressing the pen while the cursor is over the object, moving the pen, then releasing at the new position. *Touching* or *pressing* a position means tapping the pen, releasing before the pen is moved. *Lifting* means briefly lifting the pen far enough away from the tablet to cause the cursor to disappear. All of these motion quickly become natural with a little practice.

Text screen. The "alpha" screen displays other helpful information while the program is running. Normally the alpha screen is turned off so that the graphics is clearly readable. Pressing the EXECUTE key switches turns the alpha screen on and off.

Keyboard. Most of *LOG*'s advanced features are controlled by single-keystroke commands. A list of these commands is presented on the text screen for reference. A few commonly used functions are duplicated in the digitizing tablet's 16 menu boxes. Also on the keyboard is the *knob*, which, when rotated, causes the screen to scroll over a diagram which is too large to fit on the screen at once.

Beeper. *LOG* uses the sound generator in the keyboard to provide feedback at various times. A high-pitched tone generally means that the computer wants you to do something special, such as typing a word on the keyboard. A low-pitched tone means that there is an error or that *LOG* was unable to perform some function. The exact reason for the error will be displayed as a message, usually at the bottom of the screen.

Printer. If a printer which is capable of graphics is connected, *LOG* will be able to produce printed circuit diagrams.

A Sample Session

This section will illustrate a few simple features of *LOG*. It will show how to start the program and how to draw a simple circuit. Some basic editing and simulation features will also be shown.

With the computer displaying its normal command prompt, press the letter X. The new prompt, "Execute which file?", will appear. Type LOG and press ENTER. (Your particular installation may require a different command.) The computer will read in its basic library of gates, displaying a picture of each on the color monitor. When this is done, the screen will clear and the normal *LOG* menus will appear along the bottom, and the words "PAGE 1 OF 1" in the upper-right corner (see Figure 1 at end of manual). The computer is now ready to begin.

Take the pen and hold it against the surface of the tablet (without pressing down on the pen). A small arrow cursor should appear on the screen. As you move the pen, the cursor should follow. If you lift the pen away from the tablet, the cursor disappears from the screen.

Try touching the word CAT in the lower righthand menu area. Do this by moving the pen so that the cursor (arrowhead) is on top of the word CAT on the screen, then briefly pressing the pen down on the tablet. The menus disappear and the catalog of gates is shown at the top of the screen (see figure 2). Touch any part of the screen that is not near a gate to return to the normal, working screen.

In the middle of the menu area you should see seven symbols. Five of these are the traditional building blocks of digital logic, the AND, NAND, OR, and NOR gates, and the inverter. The other two are "switch" and "LED" gates which you use to simulate your circuit. The *switch* is the pointed box; the *LED* is the square object to its right. Other types of switches and indicators can be found in the catalog. These will all be described in detail in Chapter III.

To add a gate to your circuit, move the cursor on top of the desired gate, press the pen, and pull the gate into position. When you release the pen, the gate is "installed." Use this method to install a NAND gate in the middle of the drawing area. Notice that gates in circuits are light blue, whereas the gate drawings in the menu area are green.

To move a gate, just "grab" the gate, pull it into its new position, and release. Try moving the NAND gate around on the screen. Try picking it up, moving the pen all the way to the left or right edge, and letting go. When you move an object off the edge of the screen, it is deleted. This also happens if you drop it

off the top edge, or into the text portion of the menu area.

Get another NAND gate. Now try moving it on top of the OR gate in the menu area. The OR gate is replaced. You can rearrange the menu area in this way to contain any seven of your "favorite" gates.

Since there now are no OR gates on the screen, you must go to the Catalog if you need to use one. Touch the word CAT. Find the desired gate on the screen and "grab" it. As soon as you are holding the gate, the working screen reappears. You can now drop the new gate into the circuit, or into one of the seven menu area slots. To leave the Catalog screen without grabbing a gate, just tap the pen lightly.

Put a NAND gate into the drawing area. If there is anything else in the drawing area, remove it by throwing it off the edge of the screen. As soon as you release the NAND gate in the drawing area, the computer begins to simulate it. You don't see anything because there are no switches or LED's connected to it, but in fact the NAND gate is now busy NANDing nonexistent signals at a furious pace.

To see the NAND gate at work, get a switch (the right-pointed box) and install it so that the tip of the pointed part is just on top of one of the NAND gate's two input "pins." Now take another switch and connect it to the second input pin. Finally, grab an LED and install it so that the end of the NAND's output pin just touches the "shell" of the LED. If you did it right, the LED will begin to glow. The circuit should now look like Figure 3.

(If the LED fails to glow, or if the circuit does not behave as described in the next few paragraphs, try grabbing the various parts and moving them a small distance in different directions. If you look closely at the gates, you will see a tiny red dot at each connection point. The idea is to move two gates so that their red dots are exactly on top of one another. This is *not* as difficult as it sounds!)

Notice that the switches are glowing yellow, while the LED is glowing red. *LOG* displays yellow for "off," and red for "on." An LED is dark if it is not connected to any outputs at all.

To turn a switch "on," just touch the switch. Be sure not to move the pen before you release pressure, or you will move the switch instead of switching it. As soon as you touch the switch, it will turn red. Touching it again will turn it back off (yellow).

Try turning both switches on at once. The LED connected to the output "magically" turns yellow. A little experimentation shows that, in fact, the NAND gate behaves exactly like a NAND gate should. Congratulations, you have just simulated your first circuit!

To erase the circuit, you could simply grab each of the four gates in turn, and throw it off the screen. However, this is far too much work, and there must be a better way. In this case, the "better way" is called DEL.

Touch the word DEL. You will notice two immediate changes: the word DEL has turned yellow to indicate that this feature has been selected, and the cursor has changed into a strange-looking object which, on close inspection, resembles a pair of scissors. *LOG* has several specialized cursors; these are

illustrated in Figure 4.

Try touching one of the switches with the scissors. (The exact location selected by the scissors cursor is the point where the two "blades" cross.) When you touch the switch, it immediately disappears. You can delete any object (such as a gate or a wire) from the screen in this way.

Once DEL mode is activated, it remains active until you either touch a point off the edge of the screen, or lift the pen momentarily away from the tablet. You will find that with practice you can exit from DEL and similar modes with nothing but a quick flick of the wrist.

To leave the Logic Simulator and return to the Chipmunk command menu, press the CLR I/O key in the upper-righthand corner of the keyboard.

These features are really all you need to understand in order to use *LOG* to create and simulate simple circuits. Chapter II describes the capabilities of *LOG* in more detail. One last feature to remember is HELP; if you touch the HELP menu item, or press the question-mark key on the keyboard, the Chipmunk's built-in screen will display the first of twenty brief help screens. Just follow the instructions to see the rest.

Chapter II

How To Use *LOG*

Drawing Circuits

Circuits are made up of two basic kinds of objects: gates, and wires. At the end of Chapter I you learned how to add gates to a circuit. Simply find the gate, either in the menu area or in the Catalog, then "grab" it, "pull" it to the desired position, and release the gate.

Wires are used to send signals between gates. To add a wire between two points, simply touch the two points in turn. When you tap the first point, a green wire will stretch between that point and the cursor as you move around on the screen. Wires are always either vertical or horizontal; the computer automatically finds the nearest vertical or horizontal wire to the cursor location. If there is no such wire (for example, if you hold the cursor at a 45° angle away from the starting point), a small "×" will appear at the starting point and no wire will be shown.

When you touch the pen again, a wire will be drawn. The "×" will move to the new endpoint, so you can draw a long, curving wire simply by touching the three or more vertices along the wire.

To finish a wire, briefly lift the pen away from the tablet surface. This action simply unselects the last starting point touched. See Figure 5 for some examples of wiring with *LOG*.

To **move** a gate, just grab the gate, pull it to its new position, and let it go. If you move it off the screen, it will be deleted. If you move it into one of the seven menu area slots, it will become the new occupant of that slot.

Moving wires is basically the same as moving gates. Grab any point along the wire and pull. Notice that horizontal wires move only up and down, and vertical wires move only left and right. If, while moving, the pen goes beyond one or the other end of the wire, the wire will be extended. If you move the cursor all the way to the edge of the screen and release, the wire will be deleted. Note that when removing wires and gates, what counts is that the *cursor* is off the screen; part of the object may still be in the drawing area but the whole object will still be deleted when the pen is released.

If you grab one *end* of a wire and move, the wire will be extended or contracted as you move the pen. If you grab the intersection of two wires, both wires will move.

If you connect one end of a wire to the middle of another in a "T" intersection, the wires will be "**soldered**" together automatically. A small dot will appear at the intersection to show that the wires are soldered. Although this is done automatically for T-shaped intersections, it will not be done for intersections where two long wires cross each other in a "+" intersection. To solder two

crossing wires, just touch the intersection with the cursor. A small solder dot will appear. To unsolder an intersection, touch it again.

It is a good idea to avoid soldering cross-wires where possible. Use two T-intersections instead. The reason for this is that if one of the cross-wires is moved, the wires may become unsoldered and the circuit will fail to function until you resolder the wires.

All unconnected pins of gates in the drawing area are indicated by very tiny red dots. These dots are the connection points to a gate. When connecting wires to gates, be sure that the wire touches the red dot. Although there are generally lines leading from the red dots to the body of a gate, these lines are *not* considered to be wires, and do not do anything special if you connect something to them. Notice that the computer draws all wires in green, and all gates in light blue. This is to help you distinguish between lines that are wires, and lines which are actually just part of the picture of a gate.

In intersections which involve pins of gates, *LOG* will not always display solder points, even though the wires and gates are connected together. For example, if a vertical wire leads across the output pin of a NAND gate, there will be no solder point shown because one of the "wires" involved is actually part of the gate. Solder points of this type *are* shown when you make a printed copy of your circuit, but for normal editing on the screen they are omitted.

If you touch the body of a gate, either in the drawing area or in one of the seven menu area slots, the gate will "**flip**." For example, a normal, right-pointing NAND gate will flip and become a left-pointing NAND gate. The gate is functionally the same, except that it now points in the opposite direction. Other forms of flips, such as up/down flipping and 90° rotation, are not supported. One exception is the special gates INV2 and INV3 (see the "Catalog" section to find how to read these gates in from the disk), which are specialized up-pointing and down-pointing inverters.

Some gates, such as switches, can not be flipped while in the drawing area, since tapping them has some other effect (i.e., changing the state of the switch). These gates can, however, be flipped in the menu area and then moved out into the circuit. A few gates, such as TO and FROM gates and the Clock, can never be flipped.

To **delete** a gate or wire, the easiest way is simply to pick it up and "throw" it off the screen (it is not necessary to "throw" it forcefully, but it is certainly more fun). For deleting several objects, however, the DEL feature is often faster.

To use DEL, just touch the word DEL, then touch each object to be deleted. Lift the pen to leave DEL mode when you are done. Touching a gate will remove the gate; touching anywhere on a wire will remove the entire wire. Notice that for deleting and moving wires, the entire wire is always treated as one; there is no way to treat the parts of a wire between connections as separate segments.

Another way to delete a wire is to grab one end and move it into the other end. You essentially fold up the wire until there is nothing left.

A later section of this chapter, "Advanced Editing Operations," describes other ways to move and delete large groups of objects.

Large Circuits

When your circuit becomes too big to fit on the screen, there are several options. One of these options is **paging**. When you press one of the number keys "1" through "9," the screen will switch to the page that you selected. If you press the "2" key, your circuit will appear to vanish. You can now draw a new, completely independent circuit in the new page. The original circuit will continue to run, but invisibly. (For example, a digital clock circuit will continue to keep correct time even when you are working in a different page.) When you press the "1" key, you will return to your original circuit, and page two will run invisibly. Up to nine pages may be kept in this manner.

The "PAGE *n* OF *m*" indicator in the upper righthand corner of the screen indicates the current page being edited, and the highest page number used so far, respectively.

Another way to switch pages is to use the tablet menu boxes. If you move the pen into one of the 16 boxes at the top of the tablet surface, the cursor changes into a horizontal bar at the top of the screen. Touching one of the boxes 1 through 8 will switch to the corresponding page, exactly as if you had pressed the equivalent key. To switch to page 9, you must use the keyboard.

A major shortcoming of the multiple-page scheme is that there is no way to run wires back and forth between pages. This means that each page functions as a complete and independent circuit. The TO and FROM gates, described in Chapter III, provide a way to communicate between different pages.

Two other features of the Logic Simulator, called Scroll and Zoom, allow you to manipulate single drawings of virtually unlimited size. **Scrolling** is accomplished using the *knob*, the circular disk located in the upper left corner of the keyboard. Rotating the knob in either direction causes the screen to "scroll" left or right over a wider circuit. As you scroll to the right, the contents of the screen appear to move to the left. Objects that move off the edge of the screen disappear, but are not lost: they reappear when you scroll back in the other direction.

To improve performance, the computer does not actually scroll the screen until you stop rotating the knob. Instead, a white line appears and moves across the screen; the distance from the edge indicates about how far the screen will scroll when you stop. The screen is also updated if you scroll the white line all the way to the opposite edge of the screen.

To scroll vertically, hold down the SHIFT key while turning the knob. Those familiar with the CAGED text editor will find that scrolling with the knob is already natural.

You can also scroll small distances by touching one of the last four tablet menu boxes, numbers 13 through 16. These boxes scroll a small amount to the left, down, up, or right, respectively.

With enthusiastic scrolling it is possible to get lost somewhere far away from your circuit, with no way of knowing which way to scroll to get back. If this happens, just press the "H" key on the keyboard. This key will send you Home to the initial, unscrolled position.

The **Zoom** feature is controlled by the "<" and ">" keys. (These are the shifted versions of the period and comma.) When you press the ">" key, the screen will "zoom" in on the circuit. All gates and wires on the screen will appear larger than usual. You can press ">" again for even larger objects. Since zooming in toward a circuit increases the distance between closely spaced pins on gates, this may be desirable for beginners who have not yet mastered the use of the digitizing pen.

To zoom away from the circuit, press the "<" key. Everything will appear smaller on the screen. Again, there are two levels of zooming below the normal size. Although objects become so close together that it is hard to touch them, zoom-down is useful if you want to look at all of a large circuit at once.

Using the Catalog

Since there is room for only seven gates in the menu area of the screen, *LOG* provides the **Catalog** screen, which has room for up to 70 kinds of gates. When you touch the word CAT, the Catalog is displayed. To add one of the gates shown to your circuit, grab the gate and drop it in the appropriate place. If you will be using several of a particular gate, it may be faster to drop it into one of the menu area slots first. To leave the Catalog display without grabbing any gates, just tap anywhere on the screen not occupied by a gate.

In general, you will need more than just the five basic digital logic gates (Inverter, AND, NAND, OR, NOR) to build a circuit. Such things as flip-flops, counters, and shift registers are useful, as well as multiple-input gates. In fact, *LOG* has all of these things and more, sitting on the disk and waiting to be useful.

To see a **listing** of all the gates available, press the letter "L" on the keyboard. The screen will show a list of names. The first thing you will notice is a predominance of numbers starting with "74"; these are simulations of actual TTL-family chips. The other names are for built-in devices (such as SWITCH and LED), or for "generic" gates, flip-flops, etc. (In *LOG*, all objects, whether they are two-input NANDs, 256-bit RAM chips, or toggle switches, are all called "gates" for convenience.) The names in yellow are the gates which are already present in the Catalog.

If you press the "+" and "−" keys while the Library listing is being displayed, different pages of the listing will be shown. Currently, there is just one listing page. Pressing any letter key will terminate the Library listing display and return the program to the normal, editing mode.

If you want to use one of the gates on the Library screen, move the pen until the desired gate is highlighted, then press. The gate is loaded into the Catalog, and the computer returns to allow you to select more gates if desired. If you are not holding the pen over a gate name, the word "quit" is highlighted

instead, indicating that pressing the pen now will simply leave Library mode.

Another way to read read gates from the disk is the Get command, selected by pressing the letter "G" on the keyboard. The computer will ask you for the name of the gate. Type the name and press ENTER. After a brief pause a picture of the gate you selected will appear in the Catalog display. You can now grab that gate and use it in your circuit.

If all 70 catalog slots are full, the computer will check to see if any of the gates (excluding built-in devices) are not currently in use; if it finds one, that gate is removed and the new gate is put in its place. If there are no available Catalog slots, you will not be allowed to load any new gates.

Sometimes you would like to read in a whole series of related gates. For example, you might want to read all the multiple-input NAND gates into the catalog. In this case, these gates are called NAND2, NAND3, NAND4, and NAND8. One way to do this, of course, is to press "G" and type a name for each gate in turn, or touch each gate's name in the Library. A faster way is to use **wildcards**. The wildcard characters are "*," which means "any group of zero or more characters," and "?," which means "exactly one character." For example, if you type "G NAND*" you would Get every gate whose name starts with NAND. Typing "G *AND*" will read all gates whose names contain the letters "AND."

See Chapter III for descriptions of the various gates available in the Catalog and in the Library.

Advanced Editing Operations

Often you want to work with groups of gates and wires, rather than with just one object at a time. This section describes some features which help you to work with large parts of a circuit at once.

You have already seen the **DEL** feature for deleting individual objects. DEL can also be used to delete all the objects in a particular area. First, press DEL as usual. Then, move to one corner of an imaginary rectangle surrounding all the objects to be deleted, press down on the pen, and pull to the opposite corner. As you pull the pen across the area to be deleted, a rectangle will appear on the screen to show just how much will be affected. When you release the pen, all objects inside this rectangle will be removed from the circuit.

If you start the area-delete process, but then decide you don't want to go through with it, just pull the pen and rectangle all the way to the edge of the screen and release. If the pen is released at the edge, nothing is deleted.

To delete really large areas, you can use the knob to scroll around while still holding the pen down. Any sized area may be deleted in this way.

"What happens if I make a mistake?" Never fear, the gates and wires you deleted with area-DEL are not gone forever. You can put them back with a new command, called **Paste**. After using DEL to "cut" a group of objects out of a circuit, you can use this command to "paste" them back in, either in the original location, or in any location. To use the Paste command, press the "*" key on the keyboard, or tablet menu box number 11.

When you start the Paste command, the cursor changes from an arrow to a rectangle the same size as the area that was deleted. Just position the rectangle to the location desired, and press the pen. You can draw several copies of the objects by touching several locations. To leave Paste mode, lift the pen.

The Paste command also displays a dotted yellow rectangle on the screen. This rectangle represents the original location of the deleted objects. To replace the objects in their original location, just line up the cursor box over the yellow box and press.

If you hold the cursor box still for a moment without actually pressing, the computer will draw in the objects which would be added if you pressed the pen. If you move without pressing, the objects are erased and the box follows the cursor as usual. If you press first, the objects are permanently added to the circuit. This allows more exact placement when just lining up boxes won't do.

The **Copy** command is just like DEL, except that it doesn't actually delete the objects in the box you define. It simply remembers them for a later Paste command. Copy is activated by the "/" key, or by tablet menu box number 12. The Copy mode cursor looks like an arrow with an extra-long tail.

Another common operation is shifting large groups of objects around, either to make room for additions in the middle of a circuit, or to clean up excess space. *LOG* provides the **Open** and **Close** commands for this purpose. Open and Close are located on the top row of keys on the numeric keypad ("E", "(", ")", and "^"). The first two Open horizontal or vertical space, respectively; the latter two Close horizontal or vertical space.

Suppose you have built a circuit and need to add an AND gate in a certain place, but there is not enough room for it to fit. One solution is to use the Open Horizontal space command, which is located on the letter "E" key. Press this key, then define a rectangle as you would for DEL or Copy. The rectangle will contain a horizontal right-pointing arrow, signifying that when you release the pen, everything to the right of (or inside of) the rectangle will be shifted that far to the right. Things above or below the rectangle will not be affected. Wires passing through the box will be extended as necessary.

The Open Vertical space command is the same, except that things *below* the rectangle are shifted *down*. Close Horizontal and Close Vertical delete everything in the box, and shift the stuff to the right (or below) inward to take up the room. Wires passing through the box are shortened.

The Open and Close commands may not be used to shift sections of the circuit with wires coming out to the side. For example, if you use Open Horizontal to shift a group of objects to the right, there must be no wires with one end in the region being shifted, and one end above or below the shifted area. If there are, *LOG* will refuse to perform the Open operation.

These commands, like area-DEL, can be cancelled once started by releasing the pen at the edge of the screen.

Special Editing Modes

There are several design aids which help you to draw and debug your circuits. These aids include the Crosshair and Logic Probe cursors, and AUTO mode.

Pressing the space bar, or touching tablet menu box number 9, will activate the **Crosshair** cursor. The little arrow disappears and is replaced by long horizontal and vertical lines. The lines intersect to indicate the cursor position. Since the lines stretch all the way across the screen, they can be used to line up rows and columns of objects to produce a more orderly diagram.

Pressing the period key, or touching tablet menu box number 10, activates the **Logic Probe** cursor. The cursor changes into a slightly different arrow which acts as a "logic probe" when held near a wire. The lines at the bottom of the screen which separate the drawing and menu areas change color to indicate the state of the wire: Yellow means "off," red means "on," and black means unconnected, or no wire is near the cursor. The Logic Probe cursor probes only wires; pins of gates that are unconnected, or connected directly to other gates, cannot be probed.

To return to the normal cursor, just press the space bar or period key again.

If you touch the word **AUTO** in the lower-left menu area, it will change from green to yellow to indicate that AUTO mode is now active. From now on, you need only touch *near* the wire or gate pin you want; *LOG* will read your mind and move directly to that wire or pin. This is handy for beginners, but usually a nuisance for experts. Since the HP 9836C is not adept at reading minds, it sometimes chooses the wrong thing. Touching AUTO again turns the feature off.

Labelling Diagrams

Circuits which contain large numbers of switches and LED's quickly become very confusing if there is no way to tell one switch from another. This section describes two new features, ***text labels*** and ***boxes***, which help you to produce more attractive and readable diagrams.

Text labels are lines of up to 70 characters, which can be used to identify such things as inputs and outputs, important wires, and sections of the circuit. To create a text label, type the letter "T" on the keyboard. A yellow underline cursor will appear in the lower lefthand corner of the drawing area. Now type whatever characters you want to appear in the label. Use the BACK SPACE key to correct mistakes. When you press ENTER or press the pen on the tablet, the label is permanently added to the circuit. You can now grab it and move it into place as you would for a new gate.

Labels have no effect on the operation of the circuit. You can place a label over a wire or gate without affecting the simulation. This is especially useful in conjunction with Invisible mode, discussed below.

Initially, letters you type in a label will be capitals. Press CAPS LOCK to type lower-case letters. The CAPS LOCK key will remain in effect for the next label you type. CAPS LOCK during label entry is independent from CAPS LOCK at any other time while using *LOG*.

Once a label has been created, you can change it just by touching the label with the pen. A blinking cursor appears at the beginning of the label. You can move the cursor using the knob. To change part of the label, just position the cursor over the first character to be changed, and type the new characters. To insert characters, hold the SHIFT key and turn the knob to the right. This inserts spaces into the label at the current position; you can type something else over these spaces if you wish. To delete, hold SHIFT and rotate the knob to the left. The new label becomes permanent when you press ENTER or press the pen against the tablet.

Another way to enhance your diagrams is by adding **boxes**. To add a box, press the "B" key. The cursor changes into a small box-like shape. Now press down at one corner of where you want the box to be, and draw out a rectangle in the same way as you would for an area-DEL or similar operation. When you release the pen, the box becomes permanent. Boxes may be used to highlight related sections of a diagram, indicate the packaging of groups of gates, etc. Boxes are always displayed using yellow, dashed lines.

To move a box, grab an edge and pull. The entire box follows the pen, without disturbing its size or shape. To stretch or shrink a box, grab one corner and pull in the appropriate direction.

For an example of labels and boxes, see Figure 6 at the end of this manual.

Pressing the "I" key activates **Invisible** mode. All gates and wires become invisible, leaving only labels, boxes, and the LED's and switches in the circuit. Clever arrangement of wiring and gates allows a circuit to look similar to its physical counterpart's control panel. The circuit can then be operated in Invisible mode without the distractions of gates and wiring.

In invisible mode, no gates are displayed except the lighted parts of LED's, switches and other devices. LED's on CLOCKs are not shown. The lettering on numeric keypads is also displayed. All normal gates, with the exception of the toggle switches SW2 through SW4, are invisible. You are allowed to touch switches and other input devices, but you can not move or delete anything in Invisible mode. This means that a circuit may be operated by people unfamiliar with *LOG* without fear of messing it up.

Press the "I" key again to leave Invisible mode.

Another related feature is **Shift-Invisible** mode. This is turned on and off by holding down SHIFT and pressing "I." This mode is like an opposite of Invisible mode; labels, boxes, and other lettering are ***not*** displayed, but gates and wires ***are***. This may be helpful when using Zoom since text labels do not zoom gracefully.

The Option Menu

Pressing the letter "O" displays the Option menu on the small screen. The Option menu allows you to change some of the aspects of *LOG*'s behavior. The Option menu screen is organized with general information at the top, followed by several lines which you can change using the knob and blinking cursor.

The top of the screen shows the program name and version number. This manual describes Version 1.4 of the Logic Simulator.

LOG displays the current time and date at the top of the screen. If the time and date were not set previously, these will be wrong. You can set the time and date by pressing "V" from the Chipmunk main command menu; there is no way to set the time and date from within *LOG*.

The Free Memory indicator gives an approximate amount of memory available, in bytes. In general, you will never have to worry about running out of memory for any reasonably sized circuit. The number shown may be less than the true amount if you have recently deleted a lot of objects.

The lines below these contain information you can change. To change an option, first move the cursor to the appropriate line using the SHIFT and knob keys. Then, move forward or backward through the possible values using the unshifted knob. You can also press a letter key appropriate to the current line; for example, on the first line, "Y" means Yellow, and "B" means Black.

The first option you can change is the color of LED's which are off, that is, connected to a logic "low" signal. Ordinarily, LED's are red when on, yellow when off, and black when disconnected. By rotating the knob you can change this color scheme to red/*black*/black. This makes it more easy to tell at a glance which LED's are on and which are off, but it gives no way to detect disconnected LED's. This option also affects LED's embedded in switches and other devices, but not the color displayed by the Logic Probe.

The next four lines contain the default settings for the CLOCK and SCOPE devices described in Chapter III. Each device can be either Synchronous or Asynchronous, and has a speed, either High, Medium, or Low. Scopes also have a Very Low speed. Note that any scope or clock can be readjusted at any time; these options control the default values for newly created scopes and clocks.

The Quiet Operation option suppresses the Logic Simulator's usual warning and attention beeps.

The Avoid Rabbits option ensures that no rabbits will appear on the screen at any time.

To leave the Options menu, just press the "O" key again.

Saving and Loading

This section describes the commands used to save a circuit to disk for permanent storage, and to recall these saved circuits at a later date.

To **save** a circuit, touch the word SAVE in the menu area, or hold the SHIFT key and press "S." The computer asks you for the name of a file in which to save the circuit. If you name a file which does not already exist, the file will be created. If you give the name of an existing file, the previous contents of that file will be lost.

File names consist of up to nine characters (normally capital letters). The file will be saved to the current default volume. This is normally the same disk volume in which the *LOG* program itself is stored. To specify a different volume, precede the file name with a volume name and a colon: "MYDISK:MYCIRCUIT".

If for some reason the computer is unable to save your file, a message to that effect will be displayed, and the computer will wait for you to press a key before continuing. One special case is when the disk is nearly full, and the computer is unable to find a space large enough for the file. In this case, *LOG* will ask if you want to "crunch" the disk; this operation rearranges the files on the disk to provide as much free space as possible. Crunching may take up to several minutes to finish. After crunching the disk, try again to save the file. If it still fails, you will have to save the circuit somewhere else.

Circuits are saved in *pages*. If you are currently looking at page 2, for example, only the objects on that page will be saved. Multipage circuits will have to be saved in several files. The entire contents of the page are always saved, even if the screen is only showing a small portion.

To **load** a circuit which has previously been saved, touch the word LOAD in the menu area, or press SHIFT and "L." The computer will display a list of all files on the default volume which may contain saved circuits, then ask you for the name of the file you wish to load. Note that not all the files listed in the directory will necessarily contain *LOG* circuits; *LOG* simply lists all files which could possibly contain circuits.

To load a file from a different volume, you can use the two-part form described above for the Save command. The *default* volume will remain the same. To change the default volume, type a volume name and colon alone ("MYDISK:"). This volume will become the new default volume used for all further Load and Save commands.

If the current page contained any objects at the time the Load command was used, these objects will be lost.

Printing

After you have completed you circuit, you may want a printed version of the circuit diagram. This section describes the *LOG* commands for printing diagrams.

The simplest way to print is simply to press "P." This key must be pressed *twice* to avoid accidental printing. If you press any other key while the computer is waiting for the second "P," the Print command will be cancelled.

The basic Print command causes *LOG* to print a copy of everything on the screen. The printout will be exactly the same as the contents of the drawing area, except that some extra solder dots are added for a more consistent look, and the LED color scheme is always black for "on," and white for "off" or disconnected. The menu area is not printed.

If your circuit is too large to fit on the screen, you will have to use the more advanced version of Print if you want to fit everything on one page. This version of Print allows anything up to the full width of the printer, and unlimited length, to be printed.

The first step is to press the "M" key. A pair of red "printing markers" will appear on the screen, one in the upper-left corner, and one in the lower-right. Just grab the markers and move them so that they bracket the part of the diagram that you want to print. The area can be as small or as large as necessary. You may want to use the knob to scroll across the circuit while holding a marker with the pen. The only restriction is that the upper-left marker must always be above and to the left of the lower-right marker.

Once you have marked the area to print, press "P" twice. Printing may be slow, especially for very wide and complex circuits. When you print a marked area, LED's and other displays are always blank. Otherwise, the printout is mostly the same as a screen-printing.

When you are done, you can press "M" again to remove the printing markers.

Miscellaneous Features

Oscillations. Since *LOG*'s simulation of a circuit is idealized and exact, sometimes a circuit will begin oscillations that would never occur in real life because of slight differences in propagation delays or component tolerances. A simple example is the classic two-NAND flip-flop circuit. If you have a flip-flop with Set and Reset switches which are both on (i.e., not setting or resetting), this circuit will oscillate when you load a fresh copy from the disk. Both NAND gates will initially be on; they will both respond and turn each other off at exactly the same time, which will result in turning them back on. The net effect is that the LED's on the outputs will appear to flicker between on and off.

To suppress oscillations, press the "0" (zero) key. *LOG* will try to stabilize the circuit by changing the propagation delays of gates randomly throughout the circuit. In the flip-flop example, one of the NAND's will switch first, resulting in a stable state (one on, one off). *LOG* will try to stabilize for five seconds, then

give up. You may have to try several times if there are many gates in the circuit besides the ones which are oscillating.

The "0" key will, of course, be unable to suppress oscillations in circuits which *should* be oscillating, such as CLOCK devices, or oscillators built from inverters.

ON/OFF/STEP. Ordinarily, *LOG* simulates the circuit continuously, even while you are editing it. For very large circuits, this can become cumbersome, since the overhead of simulating all those gates causes the cursor's response to become sluggish. To improve performance while working with large circuits, you can turn the simulation off by touching the word ON in the menu area. The word ON should change to OFF, and the simulation should freeze.

To turn the circuit back on again for simulation, press and hold the word OFF until it changes back to ON. Circuit simulation will continue with everything exactly the same as when it was turned off. Touching any switch or other input device will also turn simulation back on.

If you just tap the word OFF, the word STEP will appear briefly, and the computer will simulate the circuit for exactly one time-unit. A time-unit is the amount of time it takes a signal to move through exactly one normal gate. For example, if you had an oscillating NAND flip-flop and turned the power off, both outputs would be frozen either on or off. Pressing STEP once would advance just long enough for the outputs to change, then return to OFF. While the circuit is off, you can use the Logic Probe to see the signals frozen in the various wires. The STEP feature can be invaluable for tracking down the weird timing bugs that often to crop up in oscillating circuits.

Fast mode. Since editing and simulation normally occur simultaneously, the circuit simulation is not as fast as it would be if all the computer's attention were devoted to just that task. Pressing the letter "F" will put *LOG* in "fast" mode, in which the computer ignores the pen, and all of the keyboard except the letter "F," and concentrates on simulating as fast as possible. Pressing "F" again will return the computer to normal. This feature is generally not very useful, since the simulator is already plenty fast.

Chapter III

The Gate Catalog

Introduction

LOG considers any object with inputs and outputs to be a "gate." This can be as simple as a switch or NAND gate, or as complex as a 4-bit counter or a RAM chip. All these objects are created, moved, deleted, and connected in about the same way. Each gate has its own characteristic picture, and its own way of responding to its inputs. This chapter describes the types of gates provided by *LOG*, and how you can use them to build up a complete circuit.

When you start the Logic Simulator, the Catalog contains only the five simplest gates (Inverter, AND, NAND, OR, NOR) and the complete set of *Devices*, which are gates posessing special properties. The Devices are always present in the Catalog, and always located in the same slots.

Some gates do not correspond to "chips" or logic elements. For example, the SW2 gate looks and acts like a conventional toggle switch. It diverts a signal to one of two connectors, leaving the other one essentially disconnected. It can pass signals in either direction, as opposed to a conventional gate, which has specifically defined inputs and outputs.

Another group of gates are the "generic" gates, which form a collection of "typical" logic design elements, such as gates, flip-flops, and counters.

Finally, there is a large library of simulators for actual TTL-family chips. These gates are designed to act as similarly to the real thing as possible. This library currently stops in the 74170's. The *LOGED* program can be used to fill in needed gates that aren't there.

Special Devices

This section describes the built-in gates which are permanently located in the top rows of the Catalog. These devices provide input from and output to a circuit, along with various other functions. Although basically the same as any other gates, devices often have special properties. These properties are described below.

Devices, like other gates, have names. Since all the devices are automatically read into the Catalog, however, you never need to worry about their names unless you are trying to modify them with *LOGED* (see Chapter IV).

GND. The Ground symbol is very simple; it provides a constant "zero" level at its output. The basic Ground points downward; you can Get the gates GND2 and GND3 from the library for grounds pointing in other directions. GND3 can be "flipped" to point to either the left or the right.

+5V. The +5V symbol produces a constant "one" level. Of course, the name "+5V" is purely arbitrary, and is no more relevant to five volts than to any other voltage. The symbols +5V2 through +5V4 are also provided. Note that separate left- and right-pointing versions are provided since flipping would reverse the lettering.

SWITCH. This gate is a basic switch. It produces a "one" or "zero" level at its output. The switch is initially off. Tapping it once will turn it on; tapping again will turn it back off. Note that since the switch responds specially to being tapped, the only way to flip it is when it is in one of the menu area slots.

PULSE. This gate is similar to SWITCH, except that it produces pulses instead of continuous levels. The pulser is normally off. Each time it is tapped, its output goes briefly high, then returns low again.

LED. This device simply displays an incoming signal as a color, either red, yellow, or black. You can connect to any edge or corner of an LED; if there are several signals coming in, the LED will display the logical OR of all of the signals. LED's dropped on top of long wires are useful as permanent logic probes.

EDGE. The Edge Detector monitors its input, looking for changes from one to zero or from zero to one. When an edge (change) is detected, the Edge Detector turns red. You must then tap the detector to reset it before it will respond to another pulse.

CLOCK. The Clock produces a steadily-changing frequency for driving sequential circuits. Clocks have several speeds and modes. Clocks are initially set for ***high-speed***, ***synchronous*** operation (see below). This default can be changed from the Option Menu. Also, you can tap a clock at any time to reconfigure it. See the end of this section for details.

SCOPE. The Scope functions as a miniature oscilloscope. It displays on its "screen" a trace showing its input signal plotted against time. If you connect a second signal to the pin on the bottom, the scope will ***trigger*** (start a new scan) every time the trigger signal changes from zero to one. If the ***trigger*** pin is left unconnected, the scope scans continuously. Scopes, like clocks, have various speeds and can be configured as described below.

7SEG. This device is a seven-segment hexadecimal display. It treats the incoming four signals as a four-bit binary number and displays a hexadecimal digit (0 through F) appropriately. The signals are arranged with the most significant bit on the bottom; this standard is universal throughout all *LOG* gates which use binary numbers. A second set of pins is provided on the bottom; these are exactly equivalent, and are provided only to give you some flexibility in laying out your circuit. If signals are present at both sets of pins, the logical OR of the bits is used. Of the bottom pins, the left pin is the most significant.

TIE. This gate functions as a pull-up resistor for use with open-collector gates, such as the TTL 7401. It causes the wire it is connected to to go "high," but *only* if nobody else is trying to pull it low.

TO, **FROM**. These arrow-shaped gates are used for communicating signals over long distances where running wires would be a nuisance (or impossible, such as wiring between pages). For example, suppose you had a clock signal which you wanted to send to various places all over a large circuit. All you need to do is

connect a TO gate (arrow pointing right) to the clock source, then tap the space immediately to the right of the arrow. A red underline cursor will appear; now type any name up to six characters long (say, "CLOCK"). Then, each time you need this signal, take a FROM gate (warped arrow) and write the exact same name in the space to its left. Signals sent to the TO gate will magically appear at all FROM gates with the same name.

If you tap a TO/FROM gate by accident, press ENTER immediately to restore the original name. When entering names, remember to press ENTER at the end of short names, but not for names that are a full six characters long (just listen to the prompting beeps). Also, like Text Labels, TO/FROM entry has a CAPS LOCK key which carries over into further TO/FROM name-entering. And remember: *LOG* imposes a limit of only 500 unique TO/FROM names per circuit!

KEYPAD. This device acts like a hexadecimal keypad. When you touch one of the symbols on the keypad, the corresponding binary number appears on the four output pins. This number remains on the outputs until a new key is pressed. The pin on the bottom is a strobe; it produces a brief high pulse every time a key is pressed.

BREAK. This device is the "hardware breakpoint." It monitors its inputs looking for low-to-high changes on the plain pin, or high-to-low changes on the pin with the bubble. As soon as one of these transitions occurs, the simulation is immediately turned "off," freezing everything for detailed analysis. This is similar to the breakpoint mechanism provided by most debugging software. To restart the circuit, press and hold the word OFF, or touch any switch.

Configuring Clocks and Scopes. Clocks and Scopes can be reconfigured at any time for various speeds and operating modes. Speeds are HIGH, MEDIUM, and LOW. Scopes also have a VERY LOW setting. For clocks, these speeds correspond to maximum speed, 10 Hz (cycles per second), and 1 Hz, respectively. Clocks and scopes also give you the choice of Synchronous operation, in which case the clock or scope will wait for the *rest* of the circuit to stabilize before continuing, or Asynchronous operation, in which case the devices operate at full speed even if, for example, other parts of a circuit are still reacting to a previous clock pulse.

To reconfigure a clock or scope, simply touch the device's body. The device will turn purple, and a list of choices will appear on the text screen. You can now press one of the letters H, M, L, or V to set the speed, or S or A for synchronous/asynchronous operation. You can also use the tablet menu boxes if you want: boxes 1 through 4 correspond to V through H speeds, and boxes 5 and 6 correspond to A and S modes, respectively.

To finish configuring a clock or scope, press the space bar key, or press the pen on the tablet.

Special "Gates"

This group includes toggle switches, and various types of jumper wires which all share the property of being able to conduct signals in either direction.

The gates **SW2**, **SW3**, and **SW4** are *toggle switches*. That is, they are switches composed of an arrow leading from one connector to one of two others. Tapping the switch flips the arrow so that it points to the alternate connector. The switch acts very much like a plain wire between the two involved connectors. See the section on "Design Considerations" for a clarification of the phrase "very much like."

The SW2 version has one connector at the bottom and two at the top, with an upward-pointing arrow. SW3 is reversed, with an arrow pointing down to one of two connectors. SW4 is a double-pole, double-throw switch which acts like a pair of SW3's that change together with a single tap.

The **JUMPER** family contain various bits and pieces of conducting wire which can be used for just about any application where you want an easily removable wire segment. These are designed to be used to help simulate jumper wires and DIP switches.

The **CROSS** and **CRUNCH** families contain bunches of wires arranged in ways that are awkward or annoying to draw out using regular wires. CROSS2 and CROSS3 act like a pair of crossing wires; CROSS4 simulates four crossing wires. CRUNCH and CRUNCH2 convert between widely-spaced pins (such as those used by 7SEGs) to closely-spaced pins (such as those on the more advanced counters, 74160-74163).

Generic Gates

"Generic" gates are general-purpose gates which are not associated with actual chips. They are best suited to simulations of abstract logic designs, as opposed to designs which are intended to be actually built.

Naturally, the simplest generic gates are the gates themselves: **AND**, **NAND**, **OR**, and **NOR**. These each come in 2-, 3-, 4-, and 8-input versions. For example, the two-input NOR gate is called NOR (this one is loaded automatically into the catalog), and the 3-input AND gate is called AND3. As usual, these right-pointing gates can be flipped into left-pointing gates just by tapping them. There are no up- or down-pointing simple gates.

A parallel group of gates, **ANDX**, **NANDX**, **ORX**, and **NORX**, are the same gates with DeMorgan's theorem applied. For example, NORX3 acts exactly like NOR3, but it *looks* like AND3 with bubbles on the inputs. Although using these gates instead of the normal ones does not affect circuit operation, it can often make the design easier to understand. The DeMorgan gates are available in 2-, 3-, and 4-input formats only.

The fifth basic food group of digital logic, **inverters**, include the INV, INV2, INV3, INV4, and INVX gates. INV is the normal right-pointing inverter. INV2 and INV3 are up- and down-pointing inverters for use in special situations. INV4 is a

group of four small, closely packed inverters. INVX is a DeMorgan inverter, with the bubble on the input instead of the output.

There are three types of **flip-flops**, each with positive or negative clock options. DPOS and DNEG are flip-flops which copy their D input to their Q output on the low-to-high (DPOS) or high-to-low (DNEG) transition of their clock input. Sending a zero to the pins on the top or bottom will clear or preset the flip-flop regardless of the clock. TPOS and TNEG are similar, except that the T input controls whether the outputs should toggle or stay the same on the next clock. JKPOS and JKNEG are traditional J-K flip-flops.

LATCH is a gated latch. When its clock input is high, the data input is passed directly on to the output. When the clock goes low, the output freezes at the most recent data value.

10COUNT and **16COUNT** are decimal and binary counters patterned after the TTL 7490 and 7493 chips, respectively. Each features a negative clock, four-bit output (bottom pin is most significant), and clear input. 10COUNT also has a preset-to-nine input.

SHIFT is a basic four-bit shift register. It has four loading inputs, four outputs, and clock, mode, and serial data inputs. On the positive transition of the clock, the gate either loads (if Mode is Low), or shifts (if Mode is High). Shifting occurs in the direction indicated by the arrow on the gate's picture. The serial data input is shifted into the first bit.

RAM and **ROM** are 256-bit memories. In fact, both are equivalent, except that the contents of ROM's are saved when a circuit is saved to the disk, whereas RAM's will not be saved, but will come up randomly next time the circuit is loaded. The gates include eight address pins on the bottom, forming an address from 0 to 255, plus data-in and data-out pins, and inverted write-enable and chip-enable inputs. When chip-enable is held high, the chip's output is essentially disconnected from the circuit, and the inputs are ignored. When chip-enable is low, the currently addressed bit appears on the data-out pin. If write-enable is also low, the value on the data-in pin is stored in the current bit, and also passed on to the output pin.

A close look at a RAM or ROM in the drawing area reveals a row of red dots along the top of the gate, matching the eight address lines below. These dots are actually hidden pins. The RAM or ROM gate continuously outputs a copy of its address input onto these pins, so that a second memory stacked above this one will also get the address. This allows wide (e.g., 8-bit) words to be formed without excessive address wiring. The penalty is that there is a one-time-unit propagation delay on the address lines, meaning that when an address appears at the bottom of a stack of 8 RAM's, it will take 8 time-units before the address has reached all the gates.

The TTL Library

The remainder of the gates in the Library are 7400's series TTL gates. They are implementations of various TTL chips as described in the Texas Instruments TTL Data Book. This is a discrete simulator, so complications like propagation delay and linear effects time are *not* simulated perfectly. In general, however, it is good design practice to avoid the kinds of situations in which propagation delay matters significantly.

Some of the gates, such as the 7401 2-input NAND, have **open-collector** outputs. *LOG* indicates this with a short diagonal bar across an output pin. Open-collector outputs are either low or disconnected, rather than low or high as with regular outputs. Since most, but *not all* TTL gates will interpret disconnected as "high," you must *always* use pull-up resistors (the built-in TIE device) on these outputs.

Another variation is the **tri-state** output. These are featured on the 74125 and 74126 gates, which look like noninverting buffers with extra "control" inputs. The control input is used to switch the output to the disconnected state, instead of acting like a normal TTL output. Pull-up resistors are not needed, provided that you can be sure that at least one of the tri-state outputs connected to a wire will always be enabled when other gates are expecting valid data on that wire.

In general, **bubbles** are used to indicate negative-logic inputs and outputs. Occasionally, however, bubbles are omitted due to complexity restrictions on the gate pictures. When in doubt, assume that the pins act as specified in the Data Book, even if they don't have bubbles.

A few gates have two versions. For example, the 74175 and 74175A are both simulations of the same chip; the 74175A simply omits some of the outputs so that your diagram won't be cluttered when you aren't using those outputs. The 74155 chip contains two slightly different data selectors; these two halves are listed as 74155 and 74155A in the Library.

One extra-special case is the 74150. This 16-input data selector is just too big to fit the size restrictions on *LOG* gates. So, the 74150 has been split into two halves. To use a 74150, Get both the 74150A *and* the 74150B from the Library, then piece the two halves together in the diagram. Because of its rather unique organization, the 74150 has a propagation delay equivalent to three normal gates.

Screen resolution makes complete labelling of all the pins of most TTL gates impossible. Instead, they have been arranged so that, given the knowledge of what the gate is supposed to do, it is possible to figure out which pins are which. For example, the 7483 is a four-bit adder. Its symbol has no lettering except for the part number, but it is easy to deduce that the two groups of closely-spaced pins on the left are the numbers to be added, that the similar group on the right is the result, with an extra pin on the right supplying carry-out, and the pin on top taking carry-in. It becomes clear when you look at the picture. When in doubt, experiment!

Design Considerations

Propagation Delay. Although *LOG* makes no attempt to simulate real gate propagation delays, there is some delay associated with each gate. In almost all cases, this delay is exactly one basic time-unit, regardless of the complexity of the gate. For example, a clock pulse to a counter will cause a change in the outputs in exactly as much time as it would take that pulse to pass through an OR gate.

Wiring has no delay, regardless of length or number of corners turned. If two pins are connected by a solid (possibly curving) line of green, then the signal will pass from end one to the other instantaneously. A few special cases are TO/FROM connections, which impose a one-gate delay for both the TO and the FROM gates (thus, a signal sent through TO and FROM will arrive two time-units after a signal sent directly by wire), and the special wiring gates such as CRUNCH and SW2, which impose a one-gate delay on the signal.

Node Conflicts. A *node conflict* is a condition in which two gates are attempting to send conflicting signals to the same place. For example, if you connect two regular NAND gate outputs to the same wire, and one of these gates turns "on" while the other turns "off," a node conflict will result on that wire. This could be a serious error in a physical circuit, and to indicate the gravity of the situation, *LOG* turns the entire wire in question blazing red until the conflict goes away.

LOG is able to keep track of only one node conflict at a time, so if several conflicts happen at once, only one of the wires will turn red. Also, if there is no wire involved (if the pins are connected directly together), the conflict will be invisible.

Chapter IV

Using *LOGED*

Simple Commands

LOGED is a utility program used to maintain the GATES file, which contains all the information about the gate library. It is *not* necessary to understand anything in this chapter in order to use *LOG*. You may want to learn to use *LOGED*, for example, if you find that you really need a particular gate or chip which the library does not provide, and are willing to add this gate to the library in order to be able to use it.

The first step is to **make a backup copy of the GATES file**. *LOGED* will gladly destroy the gate library file if you ask it to, so you want to be able to replace it if something goes wrong. It is conceivable that you would want to build your own library from scratch, but this would take a lot of effort since *LOG* will not work unless all the "built-in" devices and gates are present.

To start *LOGED*, just press "X" at the main command prompt, and type the file name "LOGED" and press ENTER. (This may be different in your own installation.) When *LOGED* is loaded, it will display a menu of available commands, and display a prompt. You may now type the *LOGED* commands necessary to create or modify gates. Commands consist of one of the words listed below, sometimes followed by a second word such as the name of a gate or file.

The first command to give is "LOAD GATES". This command loads the file GATES into memory for editing. This will take a few seconds since the library is large. When it is finished, the LOAD command will disappear and *LOGED* will give you another prompt.

The basic *LOGED* commands include:

LOAD. Load the gate definitions from the file named. If there are already gates in memory, the two libraries are merged. Name conflicts are reported as the file is loaded.

SAVE. Save all gate definitions in memory to a file. If no file is named, *LOGED* assumes the name given in the most recent LOAD or SAVE command.

LIB. Display a listing of the gates. Similar to the "L" key in *LOG*, but without the fancy "+" and "−" key features. The library listing is always kept in alphanumeric order.

GATE. Find a particular gate. If it does not exist, create it. Display a picture of the gate on the color monitor. If no gate name is given, just refreshes the picture of the current gate on the monitor.

RENAME. Change the current gate's name to the name specified. If the new name is already used, you are given the option of aborting, or removing the conflicting gate.

DELETE. Remove the gate named from the library. If no gate is named, removes the current gate.

COPY. Copy the entire definition of this gate to a new gate with the given name. If that name is already used, you are given the option of removing the old version of that gate.

NEXT. Advance to the next gate in alphanumeric order.

PRINT. Print a page containing a complete description of the gate, including large-scale picture and controlling program. You get the option to enter a single-line description of the gate.

EXIT. Leave the *LOGED* program. If you have not saved the gates since the last change, you are given the option of saving the gates. The computer then returns to the Chipmunk command menu.

The **DEF** and **DRAW** commands are discussed in the following sections.

Whenever you are working with a gate, that gate's picture is shown on the Graphics screen. However, sometimes that screen is turned off so that you can easily read the Alpha (textual) screen. To move between the two screens, press the ALPHA and GRAPHICS keys near the upper-right corner of the keyboard. *LOG*'s EXECUTE key does not work in *LOGED*.

Drawing a New Gate

The first step in creating a new gate is to create its drawing. To do this, give the command "**DRAW** *gate-name*." If the named gate does not exist, it is created. If it does exist, you will be editing the gate's current picture.

If a gate is similar in appearance to another gate, you may want to use the COPY command described above to make a copy of the similar gate, then make the changes necessary to that copy rather than starting from scratch. For example, the NAND gate pictures were created by copying the AND pictures, then adding bubbles.

To understand the elements of a gate picture, it is best to start *LOGED* and look at some of the existing gates' pictures. One example, the 74163 synchronous counter, is shown in Figure 7. There are three basic elements to the diagrams: lines, pins, and the "yellow box."

The "meat" of the drawing is made up of **lines**, or **vectors**. You draw the gate by drawing out vectors on the grid of dots on the color monitor. This grid is an exploded version of the actual picture. A normal-scale version of the picture is also displayed at the bottom of the screen. A gate's picture is made up of up to 64 vectors. The purple number in the upper-righthand corner of the screen is the number of vectors left. Be very careful when this number approaches zero, since the program may crash if you draw too many vectors.

To draw a vector, first touch one of the endpoints on the grid with the pen. A "crosshair" will appear at this point. Next, touch any other point on the grid; a vector appears between those two points. The crosshair moves to the new endpoint, allowing you to draw more vectors to form a longer, curved line. To remove the crosshair, touch the pen at any point in the empty area to the right of the grid.

Pins are the connection points to gates. Although most pins are drawn a short distance away from the body of the gate, and connected to the body with wires (e.g., the regular NAND gate), the picture of the body is totally ignored by *LOG*, so the gate can look like anything you want. The only restriction is that pins ***must*** be located on the intersections of the purple lines within the grid. It is impossible while using *LOG* to touch any other point in the grid, since *LOG* "granularizes" the cursor position to aid the user in placing objects. That is why you may have noticed that pins in groups are always separated by multiples of a universal "quantum" distance.

Each pin has a number, between 1 and 32. If your gate has 10 pins, they must be numbered 1 through 10 (although within this range they may be arranged in any order on the screen); you cannot skip over pin numbers. Pin numbers are used later when you are writing the "program" which controls how the gate interacts with its pins.

To create a pin, just move the crosshair to the position for the pin, touch the word PIN on the bottom of the screen, then touch one of the numbers 1 through 32 on the left of the screen. If you touch the "cancel" area to the right of the grid rather than a pin number, the PIN operation will be aborted.

If you use a high pin number, then delete that pin (as described below) but never re-use that number, "stray pins" may appear. For example, if you are designing your 10-pin circuit and then accidentally hit 12 instead of 10, the computer will think that there are 12 pins on the gate even if you later delete the pin. The result will be one or two stray red dots in the picture of the gate. *LOGED* provides a special magic incantation for removing stray pins: First, put the crosshair on the grid dot immediately below and to the right of the upper-leftmost dot. Now when you touch the word PIN, it will turn purple instead of yellow. Now touch the number which is the true maximum number of pins in the gate (in our example, 10). The offending stray pins will be removed.

The final part of the drawing which you must adjust is the gate's **yellow box**. The yellow box defines the area which is officially "inside" the gate. For example, when you tap a location on the screen in *LOG*, the computer checks if that location is inside any gate's yellow box. If it is, that gate is flipped or otherwise affected. If the location is not "inside" any gates, it is treated as the starting point of a wire to be drawn.

Initially the yellow box is located at the rim of the grid. You must move it inside using the MOV command discussed below. The basic rule for yellow boxes is that all pins of the gate must lie *outside* the yellow box, or else it will be impossible to draw wires connecting to them. Look at some of the existing gates in the library for examples.

The **MOV** command is used to move objects in the drawing. First, put the crosshair at the grid position containing the thing(s) to be moved. This can be one or more of the following: pins, endpoints of vectors, or corners of the yellow box. Now, touch the word MOV, then touch the new grid location to which you want to move the objects. The best way to get the hang of MOV is to experiment.

The **DEL** command deletes objects. Just put the crosshair at any grid location, then touch DEL. Everything (pins and ends of vectors) at that location will be deleted. It is impossible to delete the yellow box. After you have deleted vectors, the number-of-vectors-left indicator in the upper-right corner may show fewer vectors than there actually are. However, you can always be sure that there are *at least* as many vectors left as indicated.

The **CPY** command lets you copy another gate's picture. First touch the word CPY. Then type the name of the gate whose picture you want to copy, and press ENTER. Note that this is different from the main-level COPY command in that it copies the picture *only*; the current gate's program (if any) remains intact.

When you are finished drawing the gate, touch **END**. This will return you to the main *LOGED* command prompt.

Defining a New Gate

Once you have drawn a picture of a gate, you must use the **DEF** command to define that way it will behave. Type DEF alone to define the current gate, or follow DEF with the name of the gate to use.

Gate definitions are written in a special language called **GDL**, for Gate Definition Language. GDL programs are written using a very simple screen editor, with one instruction per line. The DEF command activates the GDL editor.

Here is a sample GDL program to define a hypothetical gate:

```
#1 = #2 NAND #3
#4 = #2 NOR #3
```

This simple gate has two input pins, numbered 2 and 3, and two outputs, numbered 1 and 4. Pin number 1 receives the NAND of the inputs; pin 4 receives the NOR. When *LOG* is simulating a circuit containing this gate, it will follow these instructions, starting at the top and working down, to see how this gate works.

To enter this definition in DEF mode, just type the two lines, pressing ENTER after each. Blank spaces are optional; *LOGED* converts all GDL statements into a canonical form as soon as they are entered. Extraneous spaces and blank lines are ignored.

To move around in the GDL program, use the up and down arrow keys above the main keyboard, or hold SHIFT and rotate the knob. To change any line, just move the cursor to that line and retype. To insert a line, press the "+" key. Press "−" to delete a line. When you are finished, press the EXECUTE key

to return to the main *LOGED* prompt.

Each gate has up to 32 pins, numbered #1 through #32, for communicating with the outside world. There are also 16 internal variables named A through P, each of which hold one bit of information. Gates that have memory (such as flip-flops and counters) use these variables to remember their internal states. Note that variables are always either on or off, whereas pins can have three states: on, off, and disconnected.

GDL uses simple statements to test and change pins and variables. The following paragraphs describe these statements one by one.

To output to a pin, write "*#n = x*" where *n* is the pin number, and *x* is any *expression*. A variable name instead of the pin number will change that variable. An expression is any pin number or variable name, or the words ONE or ZERO, or a complex expression containing the operators AND, NAND, OR, NOR, XOR, and NOT, or the special words RISE, FALL, and SAME.

The AND, NAND, OR, NOR, and XOR operators perform a logical operation on two values to produce a result. They can be strung together to act on several values ("#1 NAND #2 NAND #3 NAND #4"), and combined with other operators using parentheses to show who is done first ("(#1 AND #2) NOR (#3 AND #4)"). When acting on pins, these operators assume a disconnected pin to be ONE, unless *both* pins are disconnected, in which case the result is also "disconnected." To see how this works, run *LOG* and play with a simple AND gate.

The NOT operator converts ZERO to ONE, and ONE to ZERO. If the input is disconnected, so is the result. For example, the expressions "#1 NAND #2" and "NOT (#1 AND #2)" are equivalent. A simple use of NOT is the statement "A = NOT A", which "toggles" the value of the variable A.

The RISE and FALL operators are always followed by a pin number. They are ONE if that pin is experiencing a rising or falling transition, and ZERO otherwise. The flip-flops and counters use this to monitor their clock pins.

A variation on the "pin =" statement is "pin <". This statement creates an open-collector output. For example, the program for the 7400 is "#1 = #2 NAND #3," but the program for the 7401 is "#1 < #2 NAND #3."

The IF statements are used to allow gates to do different things in different situations. The variations are: IF, which means "if the expression is ONE or disconnected, do the following things," IFNONE, which tests for disconnected only, the corresponding IFZERO and IFONE, and IFCONN, which tests if it is ZERO *or* ONE (but *not* disconnected). A useful combination is IF NOT, which tests for ZERO or disconnected.

After the IF statement come the statements which are to be controlled by the IF, then an END statement. There must be exactly one END statement for every IF (or IF-variation) statement.

For example:

```
IF #1
  #2 = #3
END
IFZERO #1
```

```
  #2 = #4
END
```

This program simulates a 1-of-2 data selector. If pin 1 is ONE (or disconnected), pin 3 is sent on to the output pin, number 2. If pin 1 is ZERO, pin 4 is sent instead.

IF can be used with any expression, including expressions involving operators and variables.

Notice that the inside of the IF-END statement was indented in this example. *LOGED* automatically does this indentation for you. Nested IFs produce extra indentation as appropriate.

It is important to be sure never to output to the same pin twice. For example, if IF NOT had been used in place of IFZERO in the above example, then ***both*** "#2 =" statements would have been done if pin 1 was disconnected. This could create a node conflict on a wire connected to only one output!

Three more esoteric features of GDL are the SAME operator, which tests two pins to see if they are connected to the same wire (or, technically, to the same node); the "pin ==" assignment, which creates a gate output with slightly higher response time (either 0 or 1 time-units depending on who is looking at the output); and RAM, which is a "magic" statement for processing RAM and ROM chips.

Sample Definitions

This section contains a few examples of real GDL programs, with explanations. These examples are taken from the standard gate library; you can look at other gates for more examples.

NAND3. This is the generic three-input NAND gate.

```
#4 = #1 NAND (#2 AND #3)
```

When you enter an expression of multiple NANDs or NORs (such as "#1 NAND #2 NAND #3"), *LOGED* will rearrange it as shown above automatically. This form is easier for GDL to work with. When entering gate definitions, you can use either notation.

INV. Ordinarily, this gate would consist of nothing more than a "#2 = NOT #1" statement. However, it turns out that an inverter is more useful if it outputs ONE when its input is disconnected, rather than passing on the disconnectedness to its output. The following GDL program performs this function:

```
IFNONE #1
  #2 = ONE
END
IFCONN #1
  #2 = NOT #1
END
```

SW2. The difficulty with this gate (the basic toggle switch) is that is must conduct in both directions. The obvious solution, "#1 = #2" followed by "#2 = #1," doesn't work. What happens is that a signal appearing on, say, pin 1 at time zero, will appear on pin 2 (correctly) at time one. But at time two, the gate will see this signal and send it ***back*** to pin 1 again, ***ad infinitum***. An effort to solve this problem eventually leads to the following program:

```
IFNONE #1 AND #2
  A = ZERO
  B = ZERO
END
IFCONN #1
  IFZERO A
    #2 = #1
    B = ONE
  END
END
IFCONN #2
  IFZERO B
    #1 = #2
    A = ONE
  END
END
```

The two variables, A and B, tell if the gate is conducting upwards, or downwards, or neither. The first IF clears both of these flags if both pins are disconnected. Next, signals pass from pin 1 to pin 2, but ***only*** if the gate was not previously conducting in the other direction. Similar restraints are put on signals sent from pin 2 to pin 1.

74163. This gate is a sophisticated synchronous counter. It is presented as an example of a fairly straightforward, but lengthy, program.

```
E = A
F = B
G = C
H = D
IF RISE #11
  IF #1 AND #12
    A = NOT A
    IF E
      B = NOT B
    END
    IF E AND F
      C = NOT C
    END
    IF E AND F AND G
      D = NOT D
    END
  END
  IFZERO #13
```

```
    A = #2
    B = #3
    C = #4
    D = #5
  END
  IFZERO #14
    A = ZERO
    B = ZERO
    C = ZERO
    D = ZERO
  END
END
#6 = #1 AND A AND B AND C AND D
#7 = A
#8 = B
#9 = C
#10 = D
```

Variables A through D contain the current count, from 0 to 15. The first step is to record this count in variables E through H for later reference. Then, the main part of the program begins. Notice that the "IF RISE #11" statement surrounds most of the active portion of the program. Pin 11 is the clock pin; this says that the counter only changes states when the clock goes from low to high.

If the two count-enable pins, 1 and 12, are both ONE, the count is increased. The statements for counting in binary are basically a direct implementation of the traditional algorithm for synchronous counters. The 74162 decade counter contains a slightly different algorithm here, so that it will return to zero after a count of nine.

The following two groups handle the Load and Clear lines, respectively. Notice that this layout implies that Clear takes precedence over Load, which in turn wins out over counting. The 74160 and 74161, with asynchronous Clear lines, are the same except that the Clear group is moved out of the clock-testing IF.

Pin 6 is the chip's carry-out line. If the count is 15, and enable pin #1 is high, this output goes high. Finally, the internal count is sent out to the output pins, 7 through 10.

74150A & B. These two gates combine to produce a fully working 74150, 1-of-16 data selector. Each half has eight of the 16 input lines; the top half also has a strobe line, and the bottom half has the four "address" bit inputs. These gates communicate through a set of "hidden" pins, which are positioned in such a way as to connect with each other when the halves are joined. Although the details of this chip's operation are to involved to describe here, you should study the 74150 if you ever want to try the multiple-piece technique on your own gates. In particular, note the way the 74150A uses variables ***before*** they are updated according to the inputs, resulting in the one- or two-time-unit delays necessary to keep to two halves synchronized.

74150A (bottom half)

Pins 10, 11, 12, 13: Address
Pins 2-9: Data
Pin 1: Output
Pins 18, 14, 15, 16, 17, 19: Communications

```
#1 = #19 OR (J NOR #18)
I = O AND ((G AND NOT P) OR (H AND P))
I = I OR (NOT O AND ((E AND NOT P) OR (F AND P)))
J = O AND ((C AND NOT P) OR (D AND P))
J = J OR (NOT O AND ((A AND NOT P) OR (B AND P)))
J = M AND ((I AND N) OR (J AND NOT N))
A = #2
B = #3
C = #4
D = #5
E = #6
F = #7
G = #8
H = #9
M = #10
N = #11
O = #12
P = #13
#14 = M
#15 = N
#16 = O
#17 = P
```

74150B (top half)
Pins 2-9: Data
Pin 1: Strobe
Pins 14, 10, 11, 12, 13, 15: Communications

```
A = #2
B = #3
C = #4
D = #5
E = #6
F = #7
G = #8
H = #9
M = NOT #10
N = #11
O = #12
P = #13
I = O AND ((G AND NOT P) OR (H AND P))
I = I OR (NOT O AND ((E AND NOT P) OR (F AND P)))
J = O AND ((C AND NOT P) OR (D AND P))
J = J OR (NOT O AND ((A AND NOT P) OR (B AND P)))
J = M AND ((I AND N) OR (J AND NOT N))
#14 = J
```

#15 = K
K = #1

Chapter V

Using *LOGH*

Format of the *LOG* Help File

LOG looks in the file LOGHELP.TEXT to find the Help information which is shown when you touch HELP or press the "?" key. This information is organized into pages of up to 20 lines each. When Help is first activated, the first page is displayed. The "+" and "–" keys allow you to step forward or backward through the pages. Pressing the "#" key, followed by any page number, then pressing the space bar or other non-numeric key, will jump immediately to any page of Help.

LOGHELP.TEXT is a regular text file, which you can edit using CAGED. The organization of LOGHELP is very simple. Each page in LOGHELP starts with an "&" character, including the first. The characters immediately following the "&" will be shown on the first line of the screen. If more than 20 lines of text follow the "&," the extra lines will be ignored.

The last line contains a dummy page which can never be seen while using *LOG*, but which must be there in order to keep things straight. The dummy page must begin and end with an "&" character.

Sample LOGHELP.TEXT file:

```
&Logic Simulator Help File (Example)

This is the first page of the Help file.
See the second page for more useful information.
&
This is the second page of the Help file.
The top of the screen is blank; the first line of text
on this page appears on the second screen line.
&Phony last record&
```

Running *LOGH*

After you have created a LOGHELP.TEXT file, **or modified the current one in any way**, you must run the *LOGH* program before using *LOG*. Otherwise, *LOG*'s Help system may not be able to find where the pages start.

LOGH simply looks at the LOGHELP.TEXT file, reads each page (displaying page numbers so that you know it's working), and creates a second file called LOGHELPIX, which is an index file that allows *LOG* to find any page without having to read through all of LOGHELP.TEXT.

To use *LOGED*, just press "X" at the Chipmunk main command menu, then type the file name "LOGH" and press ENTER. *LOGH* runs automatically from there. When the command menu returns, the new Help file is installed and you are ready to use *LOG*.

Appendix A

Library Listing

This appendix contains a listing of all the gates currently defined in the library.

+5V	Standard tie to logic "ONE" (built-in device)
+5V2	Downward-pointing +5V
+5V3	Left-pointing +5V
+5V4	Right-pointing +5V
10COUNT	Generic decade counter w/clear, set-to-nine
16COUNT	Generic binary counter with clear
7400	2-input NAND
7401	2-input NAND, Open Collector
7402	2-input NOR
7404	Inverter
7406	Inverter, Open Collector
7407	Noninverting buffer, Open Collector
7408	2-input AND
7410	3-input NAND
7411	3-input AND
7412	3-input NAND, Open Collector
74120	Pulse synchronizer
74125	Tristate buffer, positive enable
74126	Tristate buffer, negative enable
74133	13-input NAND
74134	12-input NAND, Tristate
74136	Exclusive OR, Open Collector
74138	3-to-8 decoder
74139	2-to-4 decoder
74147	Decimal priority encoder
74148	Octal priority encoder
7415	3-input AND, Open Collector
74150A	16-to-1 selector (bottom)
74150B	16-to-1 selector (top)
74153	1-of-4 selector
74155	1-to-4 decoder, positive enable
74155A	1-to-4 decoder, negative enable
74157	Quadruple 1-of-2 selector
74158	Quadruple 1-of-2 inverting selector
74160	Decade counter, asynchronous clear
74161	Binary counter, asynchronous clear
74162	Decade counter, synchronous clear
74163	Binary counter, synchronous clear
74168	Decade up/down counter
74169	Binary up/down counter
74174	6 bit register
74175	4 bit register
74175A	4 bit register (no inverted outputs)
7420	4-input NAND

7421	4-input AND
7422	4-input NAND, Open Collector
7425	4-input NOR with strobe
7427	3-input NOR
7430	8-input NAND
7432	2-input OR
7433	2-input NOR, Open Collector
7442	4-to-10 decoder
7450	2×2 and-or-invert gates
7454	4×2 and-or-invert gates
7470	J-K flip-flop with multiple inputs
7473	J-K flip-flop with Clear
7474	D flip-flop with Preset, Clear
7475	4-bit latch
7476	J-K flip-flop with Preset, Clear
7477	4-bit latch (no inverted outputs)
7482	2-bit binary adder
7483	4-bit binary adder
7485	4-bit magnitude comparator
7486	Exclusive OR
7487	True/False/Invert elements
7490	Decade counter
7492	Divide-by-12 counter
7493	Binary counter
7497	6-bit rate multiplier
7498	4-bit 1-of-2 selector/latch
7SEG	Seven-segment display (built-in device)
AND	Generic 2-input AND
AND3	Generic 3-input AND
AND4	Generic 4-input AND
AND8	Generic 8-input AND
ANDX	Generic 2-input DeMorgan AND
ANDX3	Generic 3-input DeMorgan AND
ANDX4	Generic 4-input DeMorgan AND
BREAK	Hardware breakpoint (built-in device)
CLOCK	Clock generator (built-in device)
CROSS2	Two crossing wires
CROSS3	Two crossing wires (large)
CROSS4	Four crossing wires
CRUNCH	Wide-to-close spacing converter
CRUNCH2	Alternate CRUNCH
DNEG	Generic D flip-flop, negative clock
DPOS	Generic D flip-flop, positive clock
EDGE	Edge detector (built-in device)
FROM	Receive named signal (built-in device)
GND	Standard tie to logic "Zero" (built-in device)
GND2	Up-pointing ground
GND3	Left-pointing ground
INV	Generic inverter
INV2	Generic up-pointing inverter
INV3	Generic down-pointing inverter

INV4	Quadruple inverter
INVX	Generic DeMorgan inverter
JKNEG	Generic J-K flip-flop, negative clock
JKPOS	Generic J-K flip-flop, positive clock
JUMPER	Curved vertical jumper wire
JUMPER2	Curved horizontal jumper wire
JUMPER3	Straight horizontal jumper wire
JUMPER4	Jumper with dot on the right
JUMPER5	Jumper with dot on the left
JUMPER6	Curved jumper with no dots
KEYPAD	Hexadecimal keypad (built-in device)
LATCH	Generic latch
LED	Generic LED display (built-in device)
NAND	Generic 2-input NAND
NAND3	Generic 3-input NAND
NAND4	Generic 4-input NAND
NAND8	Generic 8-input NAND
NANDX	Generic 2-input DeMorgan NAND
NANDX3	Generic 3-input DeMorgan NAND
NANDX4	Generic 4-input DeMorgan NAND
NOR	Generic 2-input NOR
NOR3	Generic 3-input NOR
NOR4	Generic 4-input NOR
NOR8	Generic 8-input NOR
NORX	Generic 2-input DeMorgan NOR
NORX3	Generic 3-input DeMorgan NOR
NORX4	Generic 4-input DeMorgan NOR
OR	Generic 2-input OR
OR3	Generic 3-input OR
OR4	Generic 4-input OR
OR8	Generic 8-input OR
ORX	Generic 2-input DeMorgan OR
ORX3	Generic 3-input DeMorgan OR
ORX4	Generic 4-input DeMorgan OR
PULSE	Pulse generator (built-in device)
RAM	Volatile 256-bit memory
ROM	Nonvolatile 256-bit memory
SCOPE	Miniature oscilloscope (built-in device)
SHIFT	Generic 4-bit shift register
SW2	Up-pointing toggle switch
SW3	Down-pointing toggle switch
SW4	Dual down-pointing toggle switch
SWITCH	Standard switch (built-in device)
TIE	pull-up to +5V for open-collector outputs
TNEG	Generic T flip-flop, negative clock
TO	Transmitter for named signals (built-in device)
TPOS	Generic T flip-flop, positive clock
XNOR	Inverted XOR gate
XOR	Exclusive OR

Appendix B

Format of Saved Circuits

This appendix describes the format of disk files created by the Save command in *LOG*. It is intended only for readers planning to write other programs to take advantage of these files.

Saved circuits are simple text files, which can be edited with CAGED and read by Pascal programs using TEXT variables and READLN statements. The file is organized into groups of lines, each line describing one instance of a particular object.

In this appendix, the word "number" refers to an integer in the range -32768 to 32767.

The first line of a file contains two numbers separated by a space; the first number describes the total number of ***nodes*** used by the circuit. Any group of connected wires constitutes a single node; gates connected pin-to-pin also communicate through nodes. In general, a node is a part of the circuit which is electrically connected together. The second number on the first line is the number of the ***null node***, which is the dummy node number given to unconnected pins of gates.

The first group of lines describes the horizontal wires. Each line contains four numbers: the X-coordinate of the left and right ends of the wire, the Y-coordinate of the wire, and the number of the node (number zero is the first node) of which the wire is a part, respectively. Coordinates are given in *LOG* internal units, with five pixels to a unit in unzoomed mode. Coordinates are relative to (3276, 3276), which is the upper-left corner of the unscrolled screen. Coordinates increase moving downward and rightward.

The horizontal-wire list ends with a line containing any four numbers, the first of which is -9999.

The next group is the vertical-wire list. This is identical to the first list, except that the data lines contain X, upper-Y, and lower-Y coordinates. This list is terminated by a second -9999 line.

The next list describes solder points. Each line contains the X and Y coordinates of a solder point, followed by the numbers of the horizontal and vertical wires involved (respectively). Wire number zero is the first wire listed in the group. This group is terminated by another line containing -9999 followed by any three numbers.

Note that the solder-point list includes those for both cross-wires ***and*** T-intersections. If all you want is the electrical organization of the circuit, and don't really care how it looks on the screen, you can ignore this list.

Next come the text labels. Each line consists of an X coordinate, a space, a Y coordinate, a colon, and up to 70 label characters. The coordinates indicate the upper-left corner of the label. This list is terminated by the line: "-9999 -1:".

The labels are followed by a group describing boxes. Each line consists of X and Y coordinates for the upper-left corner of the box, followed by X and Y coordinates for the lower-right corner. The list is terminated by a line of four numbers beginning with -9999.

The final group describes the gates. Each gate is described by one or more lines, the first of which contains five numbers and a string. The first number is the gate code. Its absolute value indicates its location in the catalog, where the upper-left corner is one. The gate code is negative for flipped gates, and positive for regular gates. *LOG* ignores this number's absolute value unless it is either 11 or 12 (corresponding to the fixed catalog locations of TO and FROM gates).

The next two numbers in the gate header line are the coordinates of the center of the gate. Next come two scratch variables whose use is different for each gate. The string is separated from the last number by a single space, and in most cases contains the name of the gate (e.g., NAND3 or SWITCH). However, TO and FROM gates are a special case. If the gate code is 11 or 12, the string indicates the signal name, rather than the name of the gate. *LOG* uses the gate code to figure out the name of the gate itself. The first scratch variable for TO and FROM gates contains a signal identification number which may be used instead of the signal name if convenient.

The gate header line is followed by one line for each pin of the gate. These lines contain single numbers which are the node numbers for those pins. If a pin is unconnected, its number will be equal to the null node number listed at the beginning of the file. Your program must know how many pins each gate has, or it can simply keep reading until it reaches a line containing more than one number.

If the gate is a ROM (deduced by examining its gate name), the pin lines are followed by 12 lines of full-range integers, in which can be found the 256 bits which are stored in the ROM, plus some garbage bits.

The end of the file is indicated by a gate header line reading "-9999 -1 -1 -1 -1 TheEnd".

Appendix C

Summary of Commands

LOG has an abundance of commands and features. This appendix attempts to summarize them all for easy reference.

Keyboard commands. In most cases, letter commands can be given in upper or lower case. Shifted-letter commands are ***always*** entered with Shifted letter keys (and vice-versa), regardless of CAPS LOCK. *LOG* considers keys on the numeric keypad to be the same as their counterparts on the main keyboard.

?	Activate or deactivate Help system.
#	Go to any Help screen by number.
zero	Suppress oscillations, or cancel suppression.
1-9	Switch to a new circuit page.
knob	Scroll the screen (SHIFT = vertical scroll).
arrows	Same as knob or Shift-knob.
>	Zoom up (larger).
<	Zoom down (smaller).
+	Next Help or Library page.
–	Previous Help or Library page.
space	Activate or deactivate crosshair cursor.
period	Activate or deactivate logic probe cursor.
/	Enter Copy mode.
*	Enter Paste mode.
(	Open Vertical space.
)	Close Horizontal space.
^	Close Vertical space.
Execute	Toggle between text and graphics screens.
B	Create a yellow, dashed box.
DD	Create a debugging printout = wallpaper.
E	Open Horizontal space.
F	Activate or deactivate Fast mode.
G	Get a gate or gates from the library.
H	Return screen to Home (unscrolled) position.
I	Activate or deactivate Invisible mode. Gates & wires disappear.
Shift-I	Activate or deactivate Shift-Invisible mode. Labels disappear.
L	View a listing of available gates.
Shift-L	Load a circuit from the disk onto the current page.
M	Activate or deactivate printing markers.
O	Activate or deactivate Option menu.
PP	Print the screen or currently marked area.
Shift-S	Save a circuit to the disk.
T	Create a text label.
CLR I/O	Exit from *LOG*.

Touching Objects. The following objects or menu-area words perform special functions when touched.

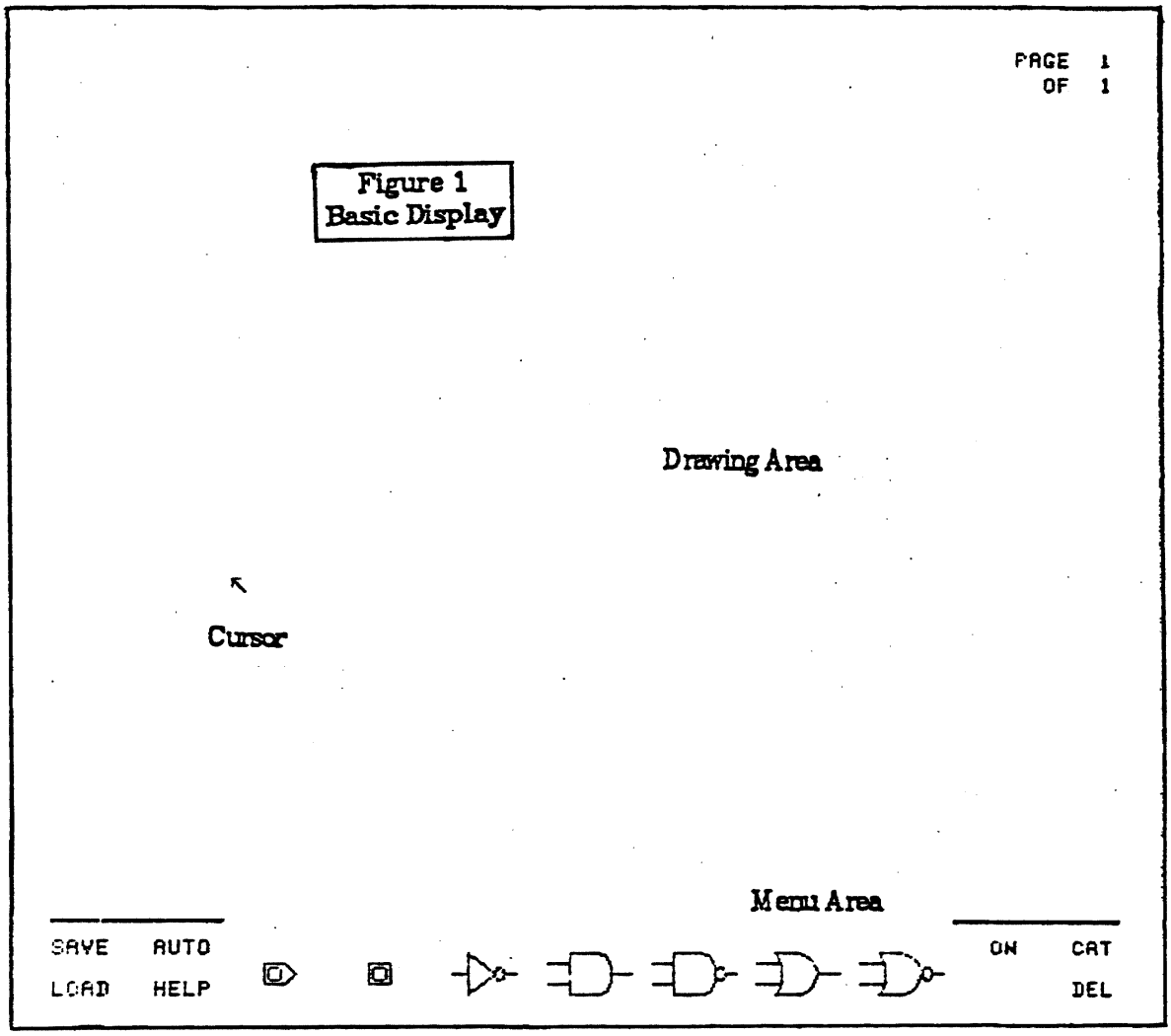
SAVE	Save the current circuit page.
LOAD	Load a circuit into the current page.
AUTO	Activate or deactivate auto-positioning feature.
HELP	Activate or deactivate the Help system.
CAT	Display the catalog of gates.
DEL	Enter Delete mode.
ON	Turn the simulation off.
OFF	Step the simulation. Hold to turn back on.
SWITCH	Turn the switch on or off.
PULSE	Send a brief pulse.
EDGE	Reset the edge detector.
CLOCK	View or change clock speed & synchronization.
SCOPE	View or change scope speed & synchronization.
TO	Enter or change the name of a signal.
FROM	Enter or change the name of the signal to read.
KEYPAD	Enter a number, between 0 and F.
Label	Edit the label.

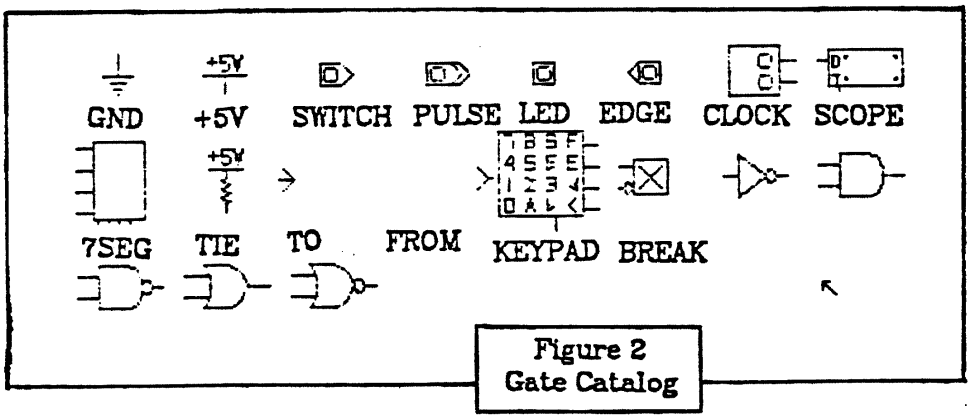
Tablet Menu Boxes. The 16 tablet menu boxes function as follows:

Box 1	Switch to page one, or select Very Low speed.
Box 2	Switch to page two, or select Low speed.
Box 3	Switch to page three, or select Medium speed.
Box 4	Switch to page four, or select High speed.
Box 5	Switch to page five, or select Asynchronous operation.
Box 6	Switch to page six, or select Synchronous operation.
Box 7	Switch to page seven.
Box 8	Switch to page eight.
Box 9	Activate or deactivate crosshair cursor.
Box 10	Activate or deactivate logic probe cursor.
Box 11	Select Paste mode.
Box 12	Select Copy mode.
Box 13	Scroll to the right.
Box 14	Scroll up.
Box 15	Scroll down.
Box 16	Scroll to the left.

Appendix D

Figures





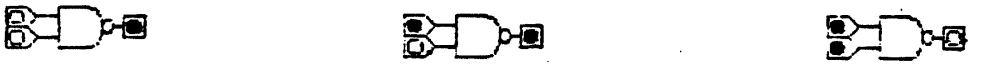


Figure 3
A Sample Circuit

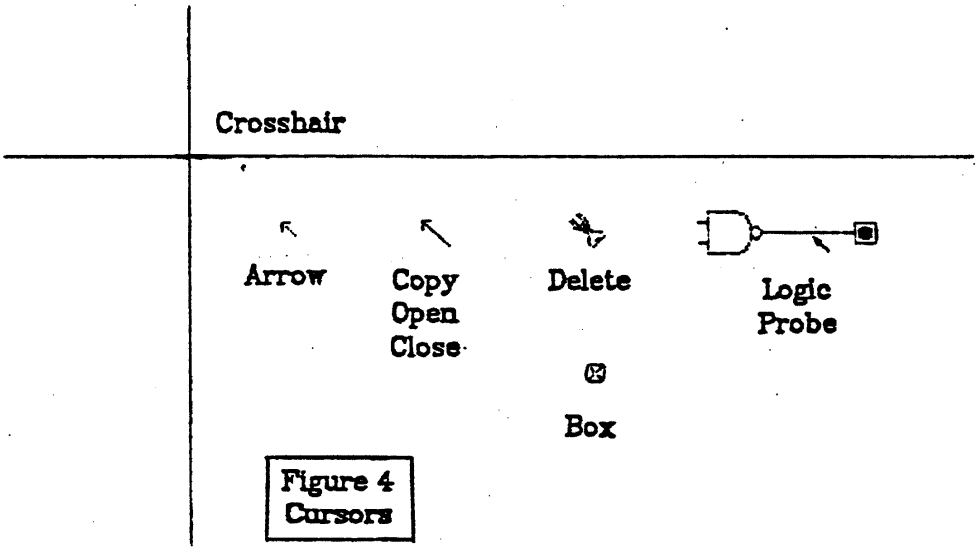


Figure 4
Cursors

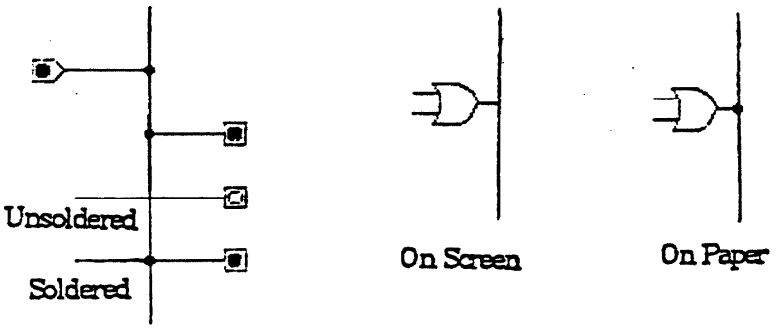


Figure 5
Wiring Examples

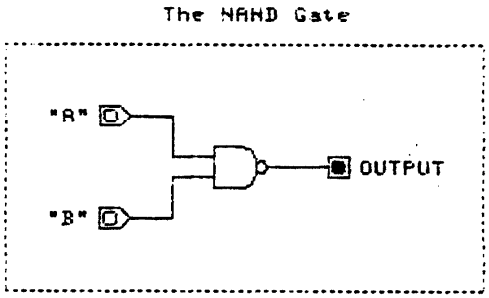


Figure 6
A Complete Circuit

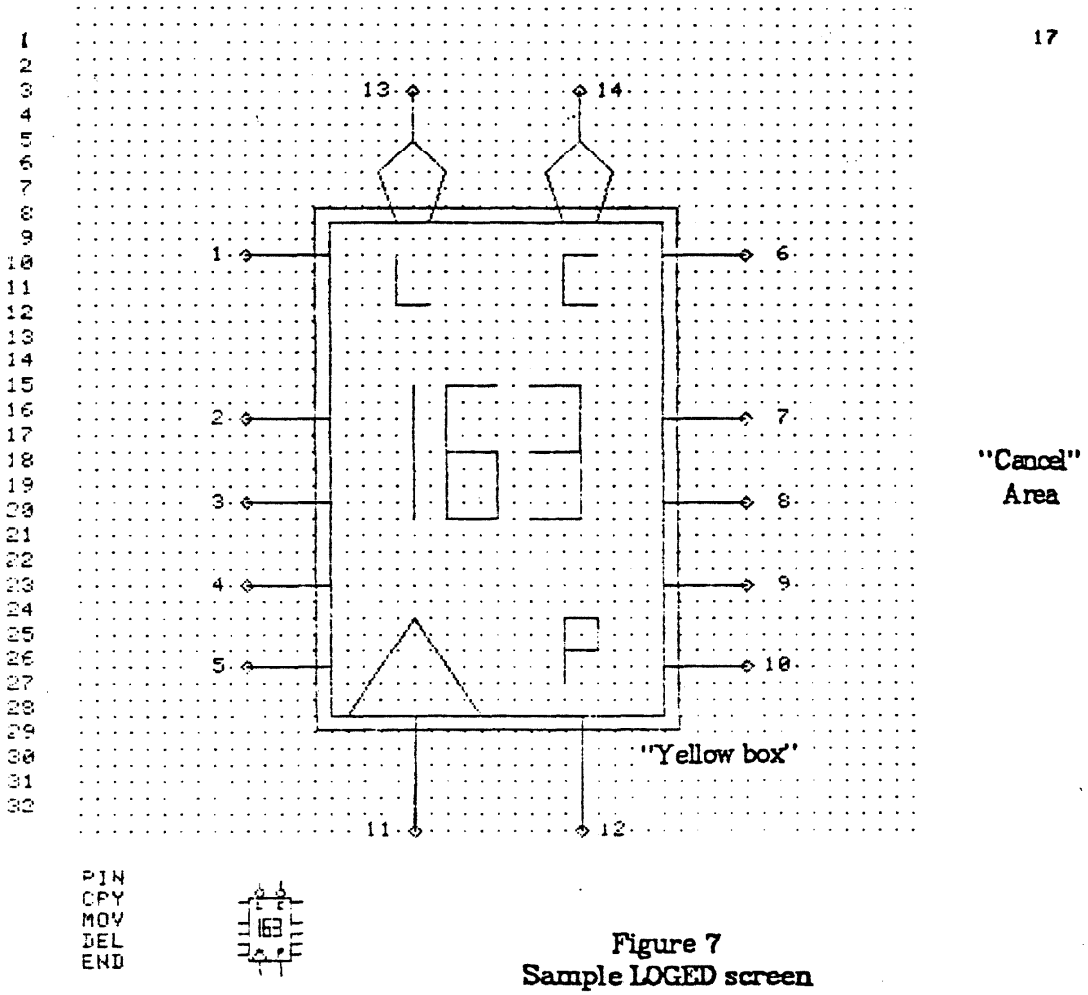


Figure 7
Sample LOGED screen